ซอฟต์แวร์
กิติพงษ์ ฤทธิบุตร

Integrated Debugger นักสืบTurbo C (2)

มาว่ากันถึงคำสั่ง
สำหรับดีบักโปรแกรม
ในเทอร์โบซี 2.0 อีกที

โลกของดีบักยังมีเรื่องราวที่น่าสนใจให้ติดตามอีกมากมายไม่รู้จบ คราวที่แล้วผู้เขียนได้แนะนำให้รู้จักกับ Integrated Debugger ที่อยู่ในเทอร์โบซีเวอร์ชัน Integrated Environment กันพอสมควร พร้อมทั้งรายละเอียดคำสั่งใช้งาน ตลอดจนหน้าที่ของแต่ละคำสั่งอย่างเป็นลำดับ วิธีการใช้งานดีบักเกอร์และขั้นตอนการดีบักโปรแกรม ในฉบับนี้ผู้เขียนจะหยิบยกสิ่งที่หลงเหลือมาเล่าสู่กันฟัง เพื่อเป็นแนวทางให้ผู้อ่านลองดีบักโปรแกรมด้วยตนเองได้ไม่ยาก งานแบบนี้ช่างท้าทายน้อยเสียเมื่อไร

Debug/Evaluate อีกที

เริ่มจากคำสั่ง Debug/Evaluate เพิ่มเติมสักนิด เนื่องจากยังมีอะไรบางอย่างที่เรายังไม่ได้เอ่ยถึง ตามที่ผู้เขียนเคยอธิบายให้ทราบว่า คำสั่งนี้จะดึงคำที่เคอร์เซอร์ทับอยู่ใน Edit window เข้ามาเก็บไว้ที่ Evaluated field โดยอัตโนมัตินั้นจะช่วยทุ่นเวลาการทำงานได้เพราะนอกจากเราจะสามารถเลื่อนเคอร์เซอร์ไปยังตำแหน่งของคำที่ต้องการก่อนเรียกใช้คำสั่งนี้ เราสามารถที่จะเพิ่มเติมข้อความลงในฟิลด์ได้อีกด้วย การใช้วิธีเลื่อนตำแหน่งเคอร์เซอร์ใน Edit window จึงสะดวกรวดเร็วกว่าการพิมพ์เพิ่มเติมเอง เช่น สมมติว่าเราต้องการประมวลผล expression 'line bfr [i-1)' ที่อยู่ภายในซอร์สไฟล์ if (linebfr [i-1] 1 = '/n') ก็สามารถทำได้เพียงเลื่อนเคอร์เซอร์ไปที่ 'linebfr' แล้วเลือกใช้คำสั่ง Debug/Evaluate เป็นผลให้ระบบแสดงคำว่า 'linebfr' ใน Evaluate field กดแป้น <Right arrow> จำนวน 5 ครั้ง (แต่ละครั้งอักขระจะเพิ่มขึ้นในฟิลด์ 1 ตัว) จะได้ '[i-1]' เพิ่มมาใน expression ของ Evaluate field แล้วจึงกดแป้น <Enter> ระบบจะทำการประมวลผลให้ต่อไป

เปลี่ยนค่า expression

คำสั่ง Debug/Evaluate ยอมให้เราเปลี่ยนแปลงค่าของ expression แต่จะต้องเป็น expression ที่ใช้ข้อมูลเพียงตัวเดียว เช่น 'linebfr [i]' หรือ '*(linebfr = i)' ซึ่งมี 'i' เป็นตัวแปรที่จะเปลี่ยนค่าได้เพียงตัวเดียว โดยระบบจะประมวลผลตัวแปร 'i' แล้วทำการเปลี่ยนค่าของมันในขั้นตอนการทำงานของดีบักเกอร์นั้นเมื่อเรากดแป้น <Enter> เพื่อสั่งให้ประมวลผลตัวแปร 'i' ดีบักเกอร์จะแสดงค่าของ 'i' ไว้ใน Result field และถ้าต้องการจะเปลี่ยนค่าก็ทำได้ด้วยการเลื่อนเคอร์เซอร์ไปที่ New Value field

โดยใช้แป้น <Down arrow> แล้วเปลี่ยนค่าตามที่ต้องการ เช่น 'i+1' (ให้เพิ่มค่า 'i' อีก 1) หรือใส่ค่าที่ต้องการเลยก็ได้ เช่น 17 และเมื่อกดแป้น <Enter> ตัวดีบักเกอร์จะประมวลผลตามค่าที่กำหนดให้ใหม่ โดยเปลี่ยนค่าของ 'i' และแสดงค่าใหม่ที่ได้ภายใน Result field ในขั้นตอนการทำงานของระบบ หากเรากดแป้น <Enter> ในขณะที่อยู่ใน New Value box จะมีผลทำให้ค่าของตัวแปรที่ใช้ในการประมวลผลเปลี่ยนแปลงไป และถ้ากดแป้น <Esc> จะเป็นการสั่งยกเลิกค่าที่เปลี่ยนนั้น เราสามารถทดสอบได้ด้วยการเปลี่ยนค่า 'i' แล้วกลับออกจากคำสั่งนี้ก่อน ค่อยเรียกใช้อีกครั้งหนึ่ง จะเห็นว่าค่า 'i' ถูกเปลี่ยนแปลงเป็นค่าใหม่ที่กำหนดให้

ขีดความสามารถของคำสั่ง Debug/Evaluate ช่วยให้เราแก้ไขข้อบกพร่องของโปรแกรมได้ด้วยการลองเปลี่ยนค่าของ expression ไปเรื่อย ๆ หรืออาจนำไปใช้ทดสอบการทำงานของ expression ในลักษณะที่มีค่าแตกต่างกันว่าจะให้ผลลัพธ์ออกมาอย่างไร เช่น ทดสอบการทำงานของฟังก์ชันบางชุดในกรณีที่มีการส่งค่าพารามิเตอร์ที่ไม่เหมาะสม แม้แต่การกำหนดค่าตัวแปรให้กับฟังก์ชันที่ต้องการทดสอบก่อนที่โปรแกรมจะเรียกใช้ฟังก์ชันนั้น ส่วนการยกเลิกคำสั่งไม่ให้เปลี่ยนแปลงค่าของ expression ทำได้ด้วยการกดแป้น <Esc> แทนการกด <Enter> เมื่ออยู่ใน New Value field เราสามารถที่จะใช้แป้น <Esc> ได้เมื่อมีการให้ค่าใหม่กับ expression โดยบังเอิญ หรือแก้ไขไปแล้วแต่อยากเปลี่ยนใจภายหลัง สำหรับการทำงานของคำสั่ง Debug/Evaluate สามารถนำมาใช้กับ C expression ใด ๆ ก็ได้ที่ไม่ใช่ expression ที่ประกอบด้วยสิ่งต่อไปนี้

- มีการเรียกใช้ฟังก์ชันอื่น
- เป็นสัญลักษณ์ประเภท defined symbol หรือ type-defined symbol หรือมาโครต่าง ๆ เช่น wordptr==0×20 จะถือว่าเป็น expression ที่สามารถนำมาใช้ได้ แต่ใช้ wordptr== SPACE ไม่ได้ถึงแม้ว่าจะมีการให้คำจำกัดความคำว่า SPACE ไว้แล้วก็ตาม
- มีตัวแปรที่ไม่อยู่ในส่วนการทำงานของฟังก์ชันที่ถูกประมวลผลในขณะนั้น นอกเสียจากจะกำหนดวิธีการเรียกใช้ไว้ (fully qualified)

ลองเรียกใช้ตัวแปร

ในการใช้ตัวแปรที่อยู่นอกขอบเขตการเรียกใช้งานของฟังก์ชันที่กำลังประมวลผลนั้นมี 2 กรณีที่สามารถทำได้ด้วยการกำหนดวิธีการเรียกใช้ชื่อตัวแปรใน expression กล่าวคือ

- เมื่อต้องการตรวจสอบค่าตัวแปรคงที่ (static variable) ต่าง ๆ ในโมดูลอื่น
- เมื่อต้องการดูผลการทำงานด้วยตัวแปรแบบออโต (local) หรือแบบคงที่ (static) ในฟังก์ชัน

สำหรับการกำหนดตัวแปรแบบเต็มรูปจะมีรูปแบบของภาษา (syntax) ดังนี้

```
<module name> . <function name>
. <variable name>
```

บางกรณีการเรียกใช้ตัวแปรแบบเต็มรูปนี้อาจจะละชื่อโมดูลหรือชื่อฟังก์ชันไว้ได้ในบางสถานการณ์ เช่น ขณะที่กำลังตรวจสอบการทำงานของ main ( ) และต้องการดูค่าตัวแปรที่เป็นแบบสแตติกชื่อ myvar ที่อยู่ในโมดูลอื่นชื่อ mysubs ก็สามารถทำได้โดยพิมพ์

```
. mysubs . myvar
```

อย่างไรก็ตาม ถ้าตัวแปร myvar เป็นตัวแปรที่อยู่ในฟังก์ชัน 'myfunc' ในโมดูล 'mysubs' จำเป็นจะต้องเรียกใช้เต็มรูปแบบ ไม่สามารถละชื่อฟังก์ชันรูปแบบที่ได้จะเป็นดังนี้

```
. mysubs . myfunc . myvar
```

ในทางตรงข้ามถ้า myfunc อยู่ในโมดูลเดียวกันกับฟังก์ชัน 'main' ก็สามารถที่จะละชื่อโมดูลไว้ ซึ่งมีรูปแบบเป็น

```
. myfunc . myvar
```

ควบคุมการแสดงผล

คราวนี้ลองทำความรู้จักกับวิธีการควบคุมการแสดงผลในวินโดว์ของคำสั่ง Debug/Evaluate (format specifier) เพราะเทอร์โบซียอมให้เราเพิ่มทางเลือกหรือรูปแบบแสดงผลให้กับ expression

ที่อยู่ใน Evaluate field ได้ โดยใช้ specifier ต่าง ๆ ที่ระบบมีไว้ให้ รูปแบบการใช้ก็คือพิมพ์ specifier ที่ต้องการต่อท้าย expression และแยกจากกันด้วยเครื่องหมาย ',' ด้วยอักษรตัวโต (uppercase) หรือตัวเล็ก (lowercase) ก็ได้

นอกจากนี้ format specifier ยังประกอบด้วยทางเลือกแบบนับซ้ำ (repeat count) ซึ่งจะต้องกำหนดด้วยเลขจำนวนเต็ม อาจตามด้วย format character หรือไม่ก็ได้ และไม่จำเป็นจะต้องมีช่องว่างระหว่างตัวเลขและ format character สำหรับ repeat count นี้จะใช้ในกรณีที่ต้องการให้แสดงข้อมูลของตัวแปรแบบต่อเนื่อง (เช่น อะเรย์) ลองดูตัวอย่างของ list เป็นตัวแปรแบบอะเรย์ของเลขจำนวนเต็ม 10 ตัว กำหนดค่าไว้ดังนี้

```
list : {10, 20, 30, 40, 50, 60,
70, 80, 90, 100}
```

ทั้งนี้ ถ้าต้องการดูเฉพาะช่วงใดช่วงหนึ่งของอะเรย์สามารถทำได้โดยกำหนดตำแหน่งเริ่มต้นไว้ที่ตัวแปรที่ต้องการ แล้วตามด้วย repeat count ที่ต้องการ เช่น ถ้าต้องการให้แสดงสมาชิกของ list ตั้งแต่ตัวที่ 6 (ตัวเลขที่เป็นดัชนีของอะเรย์จะเป็นเลข 5 เพราะเริ่มนับจาก 0) และมีค่า repeat count เป็น 3 จะได้ผลดังนี้

```
list [5], 3 : 60, 70, 80
```

จะเห็นได้ว่าวิธีการแบบนี้มีประโยชน์มากสำหรับการจัดการกับตัวแปรประเภทอะเรย์ โดยเฉพาะอย่างยิ่งอะเรย์ขนาดใหญ่ (มีจำนวนสมาชิกมาก) ที่ไม่สามารถแสดงให้เห็นได้ครบในบรรทัดเดียว

อย่างไรก็ตาม การใช้ repeat count ไม่จำกัดเฉพาะการนำมาใช้กับอะเรย์เท่านั้น แต่สามารถนำมาใช้กับตัวแปรอะไรก็ได้ รูปแบบการใช้โดยทั่วไปจะเป็น

```
var, <n>
```

อันหมายถึงการแสดงตัวแปรที่มีรูปแบบ (type) เดียวกันกับ var ที่อยู่ในตำแหน่งถัดไปอีกเป็นจำนวน n ตัว โดยเริ่มจากตำแหน่ง (address) ของตัวแปร var ค่าของ repeat count มักถูกละเลย (ไม่ประมวลผล) ถ้า expression ที่กำหนดขึ้นนั้นไม่สอดคล้องกับลักษณะหรือคุณสมบัติของตัวแปร ซึ่งมีกฎเกณฑ์ว่า สิ่งที่กำหนดขึ้นจะเป็นตัวแปรได้ก็ต่อเมื่อมันสามารถที่จะอยู่ทางด้านซ้ายของ assignment statement ได้ หรือสามารถนำมาใช้ในลักษณะอาร์กูเมนต์ของฟังก์ชันได้

อักษรชนิด format specifier

ตัวอักษรที่ใช้เป็น debugging format specifier ของ Integrated Debugger มีทั้งหมด 8 ตัวคือ C, S, D, H หรือ X, F<n>, M, P และ R โดยจะทำหน้าที่แตกต่างกันดังนี้

- C หรือคาแรกเตอร์ เป็นตัวสั่งให้ระบบแสดงอักขระพิเศษ (special display character) ต่าง ๆ สำหรับควบคุม (control character) ได้แก่ รหัสแอสกี ตั้งแต่ 0 ถึง 31 ตัวอย่างเช่น ^C จะแสดงเป็นภาพคล้ายคนยิ้ม (happy face: ซึ่งดูได้จากตารางแสดงรหัสแอสกี) คำสั่งนี้จะมีผลเฉพาะกับคาแรกเตอร์และสตริง
- S หมายถึงสตริง เป็นคำสั่งให้ระบบแสดงคอนโทรลคาแรกเตอร์ตั้งแต่ 0 ถึง 31 ด้วย

ค่าของรหัสแอสกี โดยใช้สัญลักษณ์ที่เป็น escape sequence ของภาษาซี สำหรับ 'S' specifier นี้ถือเป็นดีฟอลต์คาแรกเตอร์ของระบบอยู่แล้ว เพราะฉะนั้นมันจึงมีประโยชน์เฉพาะในกรณีที่ใช้งานร่วมกับ 'M' specifier

- D ได้แก่ decimal เป็นคำสั่งที่ทำให้ระบบแสดงค่าของเลขจำนวนเต็มออกมาในรูปแบบของเลขฐาน 10 ซึ่งมีผลต่อการแสดงผลของ expression หรืออะเรย์และโครงสร้างที่ประกอบด้วยเลขจำนวนเต็ม
- H หรือ X เป็น specifier สำหรับ Hexadecimal ซึ่งถ้าใช้คำสั่งนี้จะส่งผลให้ระบบแสดงค่าของเลขจำนวนเต็มทั้งหมดออกมาในรูปแบบของเลขฐาน 16 โดยจะมี '0x' นำหน้าตัวเลข และจะมีผลในการใช้เช่นเดียวกันกับ 'D' specifier
- F<n> หมายถึง floating-point ส่วน n จะแทนเลขจำนวนเต็มระหว่าง 2 ถึง 18 เพื่อระบุจำนวนจุดทศนิยมที่จะให้แสดง สำหรับคำสั่งนี้จะใช้ได้เฉพาะกับค่าที่เป็นทศนิยม

(floating-point) เท่านั้น

- M หมายถึง Memory dump ทำหน้าที่สั่งให้ระบบแสดงข้อมูลในหน่วยความจำ โดยเริ่มต้นแสดงข้อมูลตั้งแต่แอดเดรสแรกที่ใช้เก็บข้อมูลของ expression ที่กำหนดให้ ซึ่งมีคุณสมบัติที่จะอยู่ทางด้านซ้ายของ assignment statement ได้ ถ้าไม่เป็นไปตามนี้ ระบบก็จะไม่สนใจ (ignored) กับ 'M' specifier ที่กำหนดไว้ ในการทำงานของระบบแต่ละไบต์ของตัวแปรจะแสดงผลเป็นเลขฐาน 16 จำนวน 2 ตัว เมื่อเพิ่ม 'D' specifier เข้าไปด้วยจะทำให้ระบบแสดงผลออกมาเป็นเลขฐาน 10 แทน ในทำนองเดียวกันถ้าใช้ 'H' หรือ 'X' specifi ให้ระบบแสดงค่าแต่ละไบต์ด้วยเลขฐาน 16 ส่วน 'C' หรือ 'S' specifier ตัวแปรจะแสดงค่าเป็นสตริง (ซึ่งอาจจะมี special character ร่วมด้วยหรือไม่ก็ได้) สำหรับค่าดีฟอลต์ของจำนวนไบต์ที่จะแสดงนั้นขึ้นอยู่กับขนาดของตัวแปรเอง แต่อาจจะนำ repeat count มากำหนดจำนวนไบต์ที่ต้องการได้
- P หมายถึงพอยน์เตอร์ เป็นคำสั่งให้ระบบแสดงพอยน์เตอร์ในรูปแบบของ 'sig:ofs' และจะแสดงข่าวสารข้อมูลของตำแหน่งหน่วยความจำที่ตัวพอยน์เตอร์นั้นชี้ไป นอกเหนือจากรูปแบบของ seg:ofs ที่เป็นดีฟอลต์ของระบบฮาร์ดแวร์ ช่วยให้เรารู้ทั้งตำแหน่งของเซกเมนต์ว่าอยู่ในส่วน (region) ไหนของหน่วยความจำ รวมทั้งชื่อของตัวแปรที่ออฟเซตแอดเดรสนั้นด้วย สำหรับช่วงของหน่วยความจำที่จะใช้ประโยชน์ในการทำงานของเทอร์โบซีนั้นจะเป็นไปตามที่แสดงไว้ในตารางที่ 1
- R หมายถึง Structure/Union ใช้เป็นคำสั่งให้ระบบแสดงชื่อฟิลด์พร้อมกับค่าที่อยู่ในฟิลด์นั้น เช่น {x:1, y:10, z:5} โดยคำสั่งนี้จะมีผลเฉพาะกับตัวแปรแบบสตรัคเจอร์และยูเนียนเท่านั้น

ในเรื่องของการใช้ format specifier มีกฎเกณฑ์ทั่วไปอยู่ 2 ประการ ซึ่งหมายความว่าถ้านำ specifier เหล่านี้ไปใช้ในแบบที่ไม่อยู่ในกรณีดังกล่าวแล้ว ระบบจะไม่ทำงานตามต้องการ อันได้แก่

- การใช้งาน format specifier จะบังเกิดผลก็ต่อเมื่อได้รับคำสั่งที่ต้องการต่อจากตัวแปรที่เหมาะสม (มีรูปแบบที่สอดคล้องกัน) ถ้าไม่เป็นไปตามนี้ ระบบจะไม่สนใจคำสั่งที่ส่งให้ ข้อควรจำถ้า expression ที่ประมวลผลอยู่นั้น

แสดงผลข้อมูลหลายรูปแบบ (เช่น สมาชิกของตัวแปรแบบโครงสร้าง) เราสามารถที่จะเพิ่ม format specifier ได้หลายตัว ซึ่งระบบจะนำ specifier ที่เหมาะสมกับตัวแปรแต่ละตัวไปใช้ ตัวอย่างเช่น ถ้าเราพิมพ์

```
struct, F5H
```

เพื่อให้ระบบแสดงตัวแปรแบบโครงสร้างที่มีสมาชิกทั้งที่เป็นเลขจำนวนเต็มและเลขจำนวนจริง ระบบจะแสดงตัวเลขที่เป็นจำนวนเต็มด้วยเลขฐาน 16 (H) และแสดงเลขจำนวนจริงแบบทศนิยม (floating-point) 5 ตำแหน่ง (F5)

- ในกรณีที่มีการใช้ format specifier มากกว่าหนึ่งตัวกับ expression ตัวใดตัวหนึ่ง ระบบจะเลือกใช้เพียงตัวเดียว ซึ่งตัวที่นำไปใช้นั้นจะเป็นคำสั่งที่มีลำดับความสำคัญสูงสุดสำหรับตัวแปร

Memory Region	Evaluate Message
0000:0000-0000:03FF	Interrupt vector table
0000:0400 - 0000:04FF	BIOS data area
0000:0500 - Turbo C	MSDOS/TSR'S
Turbo C-User Program PSP	Turbo C
User Program PSP	User Process PSP
User Program-TOP of RAM	Name of a static user variable if its address falls inside the variable's allocated memory; otherwise nothing
A000:0000 - AFFF:FFFF	EGA video RAM
B000:0000 - B7FF:FFFF	Monochrome Display RAM
B800:0000 - BFFF:FFFF	Color Display RAM
C000:0000 - EFFF:FFFF	EMS Pages/Adaptor BIOS ROM'S
F000:0000 - FFFF:FFFF	BIOS ROM's

ตารางที่ 1 แสดง memory region และการใช้ประโยชน์ของหน่วยความจำในส่วนนั้น

ในระบบ (type) นั้นหากเกิดกรณีเช่นนี้กับตัวแปรประเภทสตรัคเจอร์หรือยูเนียน (เพราะมีสมาชิกหลายตัวได้) ส่วนตัวแปรประเภทอื่น เช่น ตัวแปรเดียวหรืออะเรย์นั้น การใช้ format specifier จะใช้เพียงตัวเดียวอยู่แล้ว ลำดับความสำคัญของ format specifier กับตัวแปรแต่ละประเภทได้แสดงไว้ในตารางที่ 2

เราลองมาดูวิธีการใช้ format specifier ต่าง ๆ เหล่านี้โดยใช้โปรแกรมตัวอย่างสั้น ๆ ที่แสดงการใช้ตัวแปรหลายแบบ เช่น ตัวแปรเดี่ยวหรือตัวแปรแบบอะเรย์ และตัวแปรแบบโครงสร้าง ดังตัวอย่างในรูปที่ 1

```
/* Example */
   struct {
      int account,
      char name [10];
} client = {5000, "Jones"}
int list[5] = {0, 10, 20, 30, 40};
char *ptr = list;
void main ( )
{
}
```

รูปที่ 1 โปรแกรมตัวอย่างแสดงการใช้ตัวแปรหลาย ๆ แบบ

เมื่อเสร็จแล้วให้ลองเติม expression หลาย ๆ แบบลงใน Evaluate field ซึ่งผลลัพธ์ที่ได้จะปรากฏอยู่ใน Result field สำหรับตัวอย่างของเราจะเป็นไปตามที่แสดงไว้ในตารางที่ 3

หยุดพักครึ่งทาง

เรื่องราวของ Integrated Turbo C Debugger ที่กล่าวมาทั้งหมดนี้เป็นเรื่องทั่ว ๆ ไปของ Integrated Debugger ซึ่งรวมถึงวิธีการใช้งานดีบักเกอร์ ขั้นตอนการดีบักโปรแกรม นอกจากนี้เรายังได้ลงมือดีบักโปรแกรมกันบ้างแล้ว โดยใช้คำสั่งต่าง ๆ ที่จำเป็นต้องรู้จักและใช้งานเพื่อหาบักในโปรแกรมตัวอย่าง ผู้เขียนเห็นทีจะต้องหยุดไว้เพียงแค่นี้ก่อน เพราะเนื้อหายังมีอีกมากพอควรทีเดียว ในฉบับต่อไปเราจะกลับมาหาบักกันใหม่ ผู้เขียนจะได้กล่าวถึง breakpoint เพิ่มเติม รวมทั้งการใช้งาน Watch window ตลอดจนเรื่องราวเกี่ยวกับ Watch expression การควบคุมการทำงานของ Window แล้วลองค้นหาบักที่เหลือกันอีกสักหน่อย แถมท้ายด้วยแนวทางการทดสอบซอฟต์แวร์โดยใช้ดีบักเกอร์เป็นเครื่องมือ

อย่าลืมติดตามให้ได้นะครับ

หมายเหตุ ผู้สนใจสามารถหาซอฟต์แวร์นี้ได้ที่ร้านบุ๊คเซสท์ สยามสแควร์ ซอย 2

Type	Specifiers in order	Default
char	C S H D	S
unSigned char	C S H D	S
int	H D C* S*	D
unsigned int	H D C* S*	D
long	H D C* S*	D
unsigned long	H D C* S*	D
char ptr	C S P H	S
other ptr	P H	P
enum	H D C* S*	D (followed by member name)
float	Fn	F7
double	Fn	F15
long double	Fn	F18
array of char	C S H D	S
other array	elements enclosed in braces { } and separated by commas	
structure	R	

* character type format specifiers will be accepted only for values that fall within the appropriate range (-128 to 127 for 'signed' types and 0 to 255 for 'unsigned' types)

Note : the H format specifier used by itself with a pointer variable displays the pointer as a hexadecimal integer value.

ตารางที่ 2 แสดงลำดับความสำคัญของ format specifier และค่าดีฟอลต์ของตัวแปรแต่ละประเภท

Evaluate	Result
list	{0, 10, 20, 30, 40}
list [2], 3	20, 30, 40
list [2], 3x	0×14, 0×1E, 0×28
list, m	00 00 00 0A 00 14 00 1E 00 28 00
ptr	DS : 0198
ptr, P	DS : 0198 [-list]
*ptr, 3	0, 10, 20
client	{5000, "Jones \0\0\0\0\0"}
client, r	{account : 5000, name : "Jones \0\0\0\0\0"}

ตารางที่ 3 แสดงผลลัพธ์ที่ได้หลังจากเติม expression หลาย ๆ แบบลงใน Evaluate field